

TRACK 2

JAVA CONCURRENCY PRIMITIVES

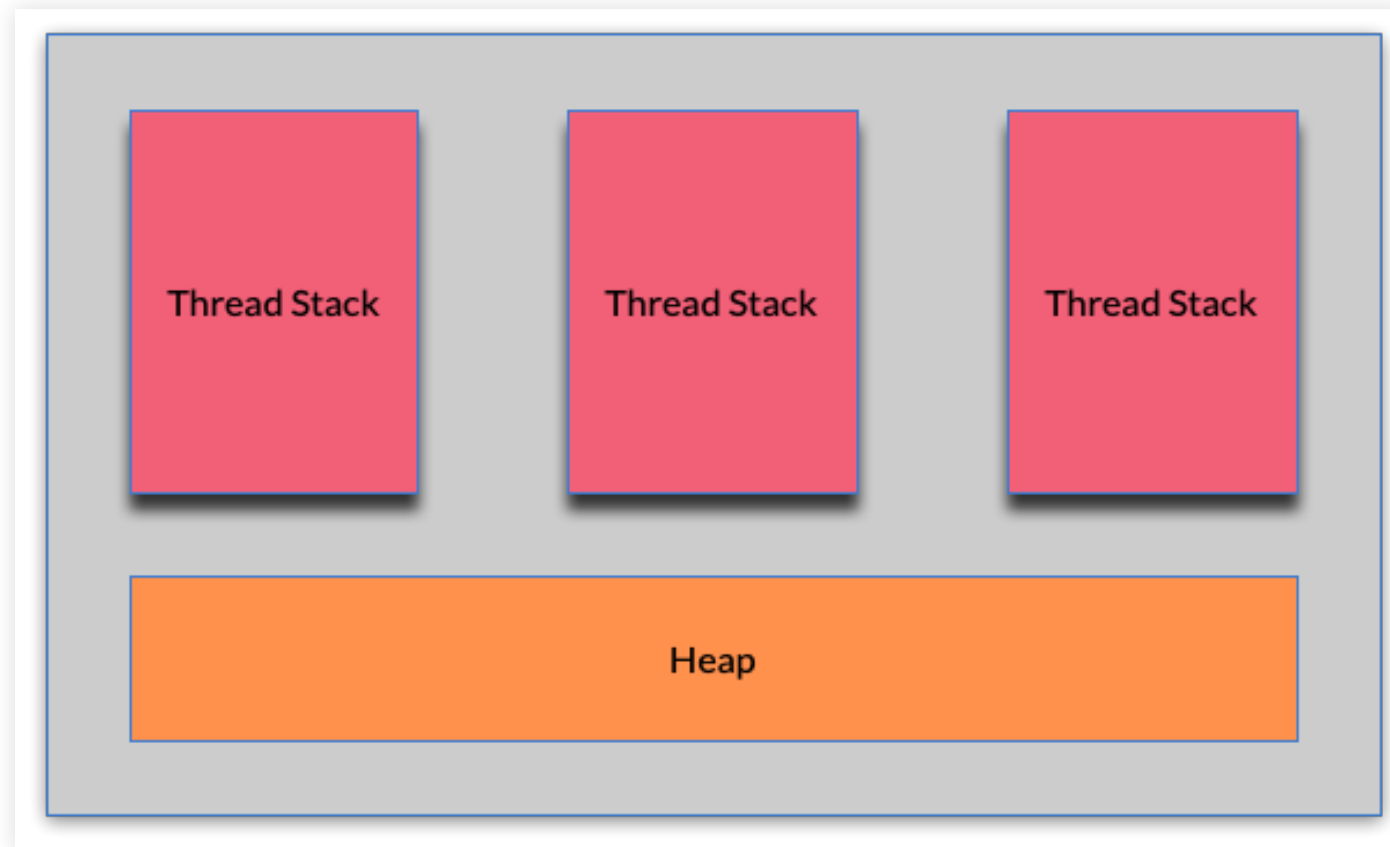
Daniel Hinojosa

JAVA MEMORY MODEL

ABOUT THE MEMORY MODEL

- Still the same Memory Model since Java 5
- Comes in two components
 - Stack (or Thread Stack)
 - Heap

MEMORY DIAGRAM



THREAD STACK

- Every thread in the JVM has its own thread stack.
- Thread stacks contains information about what methods the thread has called to reach the current point of execution.
- Referred as the "call stack", which if called repeatedly results in `StackOverflowError`
- As the thread executes its code, the call stack changes.

THREAD STACK INTERNALS

- Thread stack also contains all local variables for each method being executed internal to the stack
- A Thread can only access the local variables in it's stack
- Local variables in a Thread are not visible to another Thread
- Each thread will only have access to it's copy, but copies can be sent to other threads

JVM HEAP

- Contains the objects that are used by the Threads in the Stack
- Objects also include wrappers `Integer`, `Long`, etc.

JVM PRIMITIVES AND OBJECTS

- If the variable is a primitive type, then it is stored in the Thread stack
- Object References
 - If the variable is a object reference, the variable is stored in the Thread Stack
 - The object that the object refers to is stored in the Heap and undergoes heap lifecycle
 - The object's member variable are also stored on the heap
 - `static` class variables are stored on the heap with the class definition

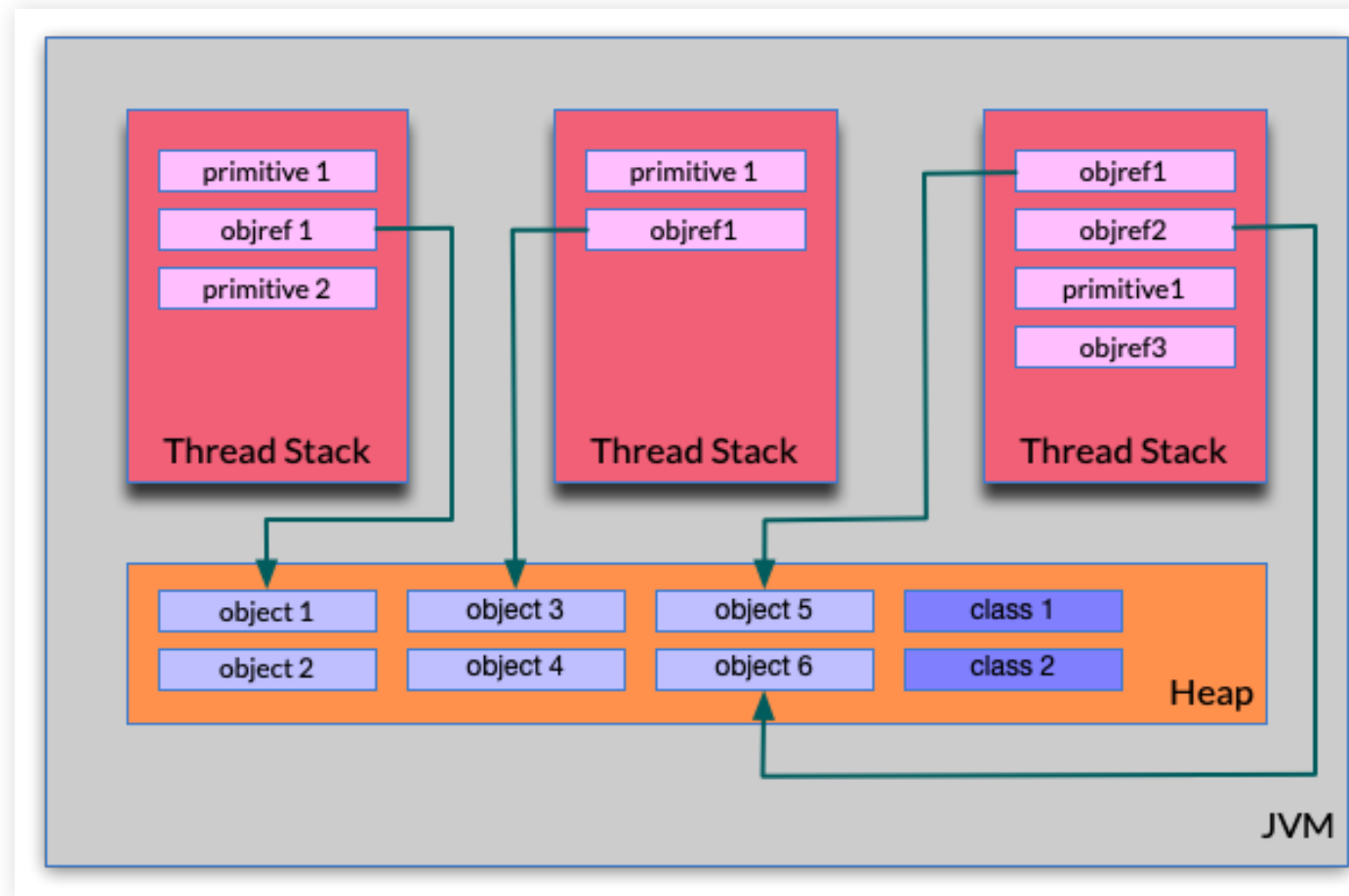
HEAP OBJECTS

- Objects on the Heap can be accessed by any Thread Stack
- Two references on the Thread Stack can access:
 - Object reference
 - Object's class reference
 - Object's member variables
 - Object's class `static` members

MEMORY MODEL AND PHYSICAL MEMORY

- Hardware memory architecture does not distinguish between thread stacks and heap
- Both the thread stack and the heap are located in main memory
- Parts of the thread stacks and heap may sometimes be present in CPU caches and in internal CPU registers

JAVA MEMORY DETAILED



OBJECTS AND GARBAGE COLLECTION

- Once objects are never used in the Heap they are *Garbage Collected*
- Occasionally, from the JVM
 - Memory is reclaimed by deleting garbage
 - Memory is compacted by *defragmenting*

HEAPS ARE BROKEN INTO SUBHEAPS

- Newly Created Objects (**Age = 0**)
 - Allocated to a young area: Eden
- New Objects that live beyond the first GC cycle (**Age = 1**)
 - Move to Young Area (Survivor S1)
- Objects that survive a few GC Cycles (**Age = ++**)
 - Gets moved between young areas: $S1 \rightarrow S2$, $S2 \rightarrow S1$
- Objects that survive longer (**Age > 15**)
 - Gets moved to an older area: Tenured
- Method definitions & `static` declarations
 - Allocated to *PermGen*, need not be allocated

VIRTUAL STACKS AND HEAP AND PHYSICAL RAM

ISSUES WITH THE JVM

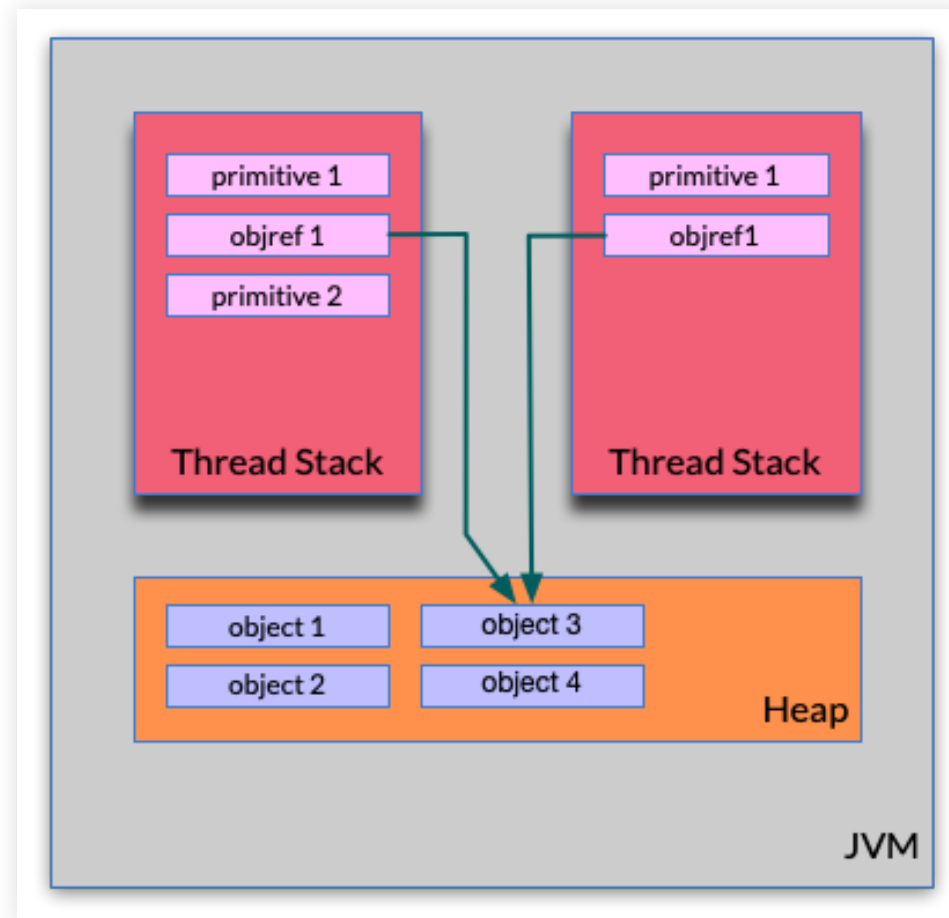
There are two issues in regards to physical memory when running the JVM

- Visibility of thread updates (writes) to shared variables.
- Race conditions when reading, checking and writing shared variables.

VISIBILITY OF SHARED OBJECTS

- If two or more threads share an object, without proper mitigation, updates to the shared object made by one thread may not be visible to others
- Proper mitigation techniques:
 - `volatile`
 - `synchronized`

SHARED OBJECTS



RACE CONDITIONS

- If two or more threads share an object
- More than one thread updates variables in that shared object race conditions may occur.
- Analogy:
 - You are planning to go to a movie at 5 pm.
 - You inquire about the availability of the tickets at 4 pm.
 - The representative says that they are available.
 - You relax and reach the ticket window 5 minutes before the show.
 - I'm sure you can guess what happens: it's a full house.

Source: <https://stackoverflow.com/questions/34510/what-is-a-race-condition>

HAPPENS BEFORE GUARANTEE

TO UNDERSTAND HAPPENS BEFORE

- The CPU can determine what can split in parallel by the CPU
- CPU can decide how to parallelize called *instruction reordering*
- May process a look-ahead, in other words, a and f can be evaluated first

```
a = b + c
```

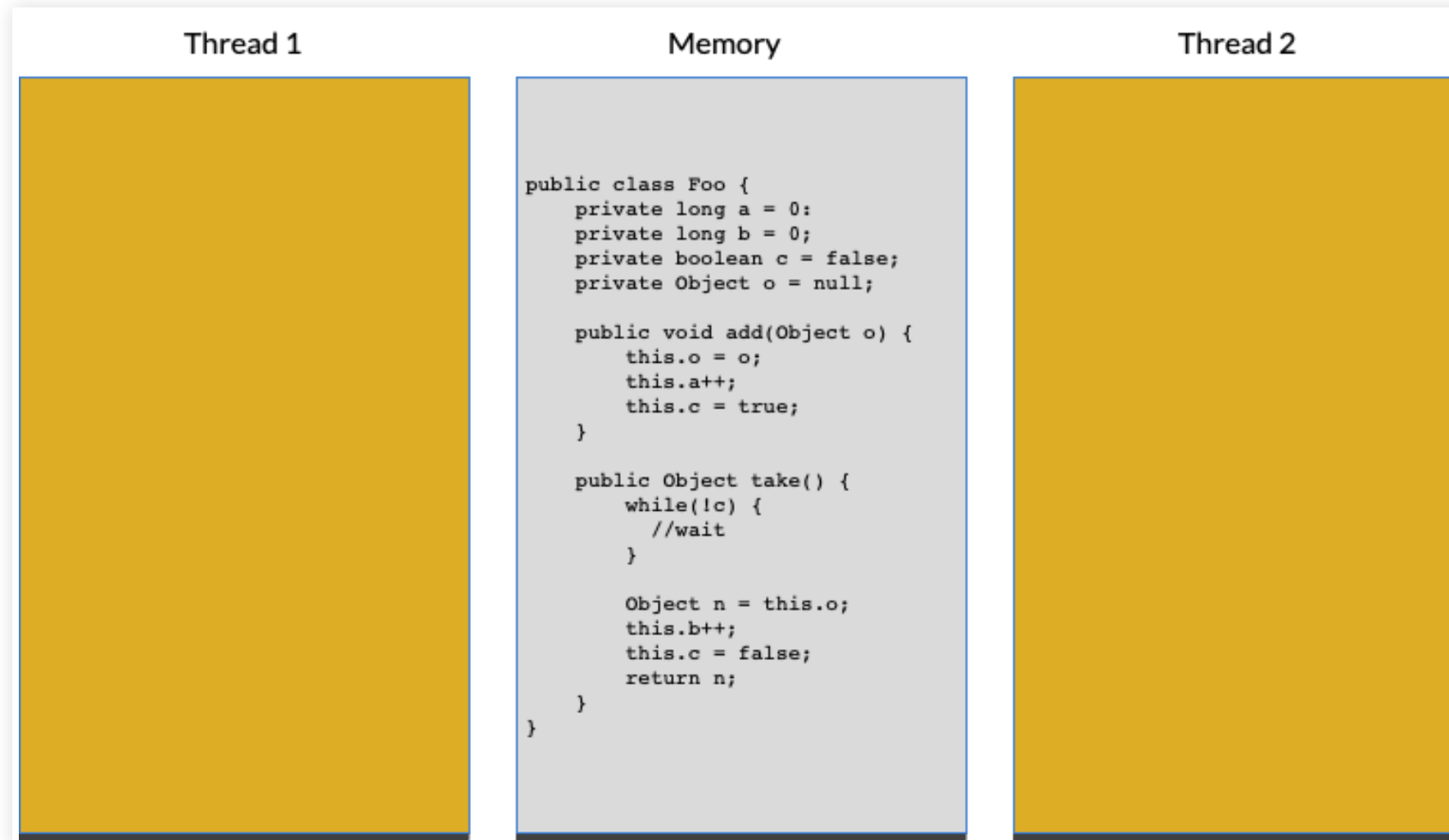
```
d = a + e
```

```
f = g + h
```

```
i = f + j
```

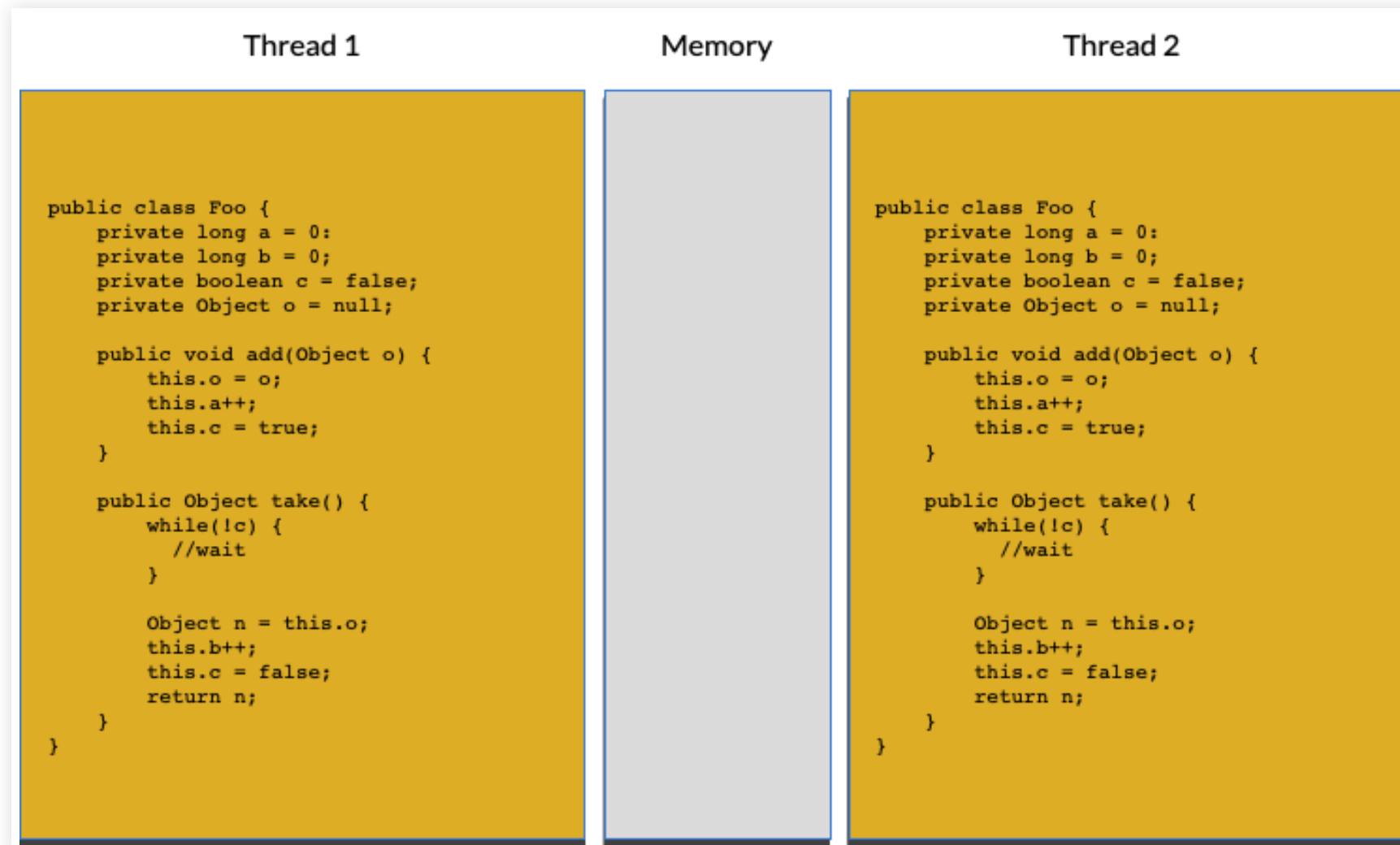
HAPPENS BEFORE GUARANTEE

The member variables are stored in memory



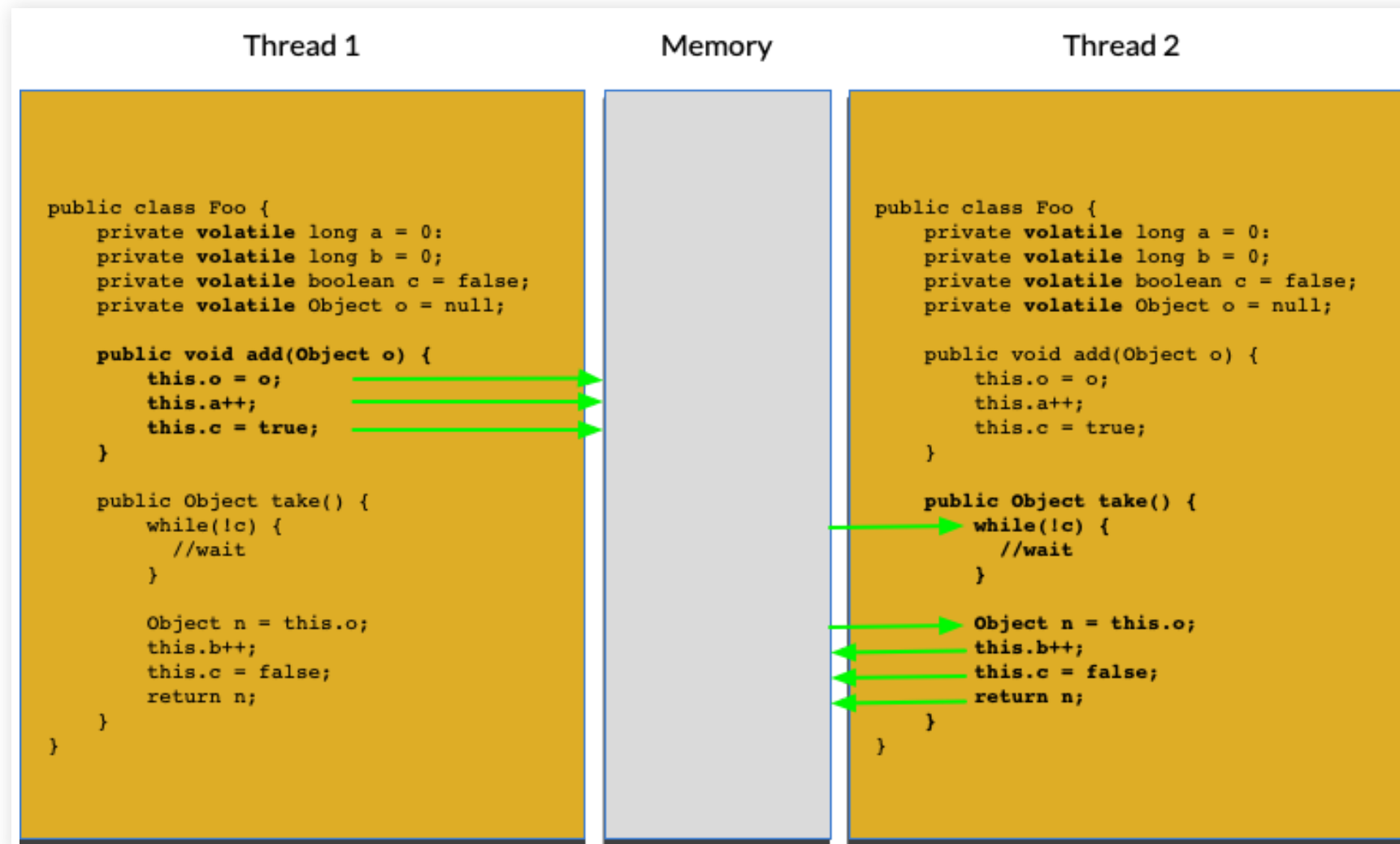
CPU CACHE ISSUE

Each thread may have their own copies in either registers or cache

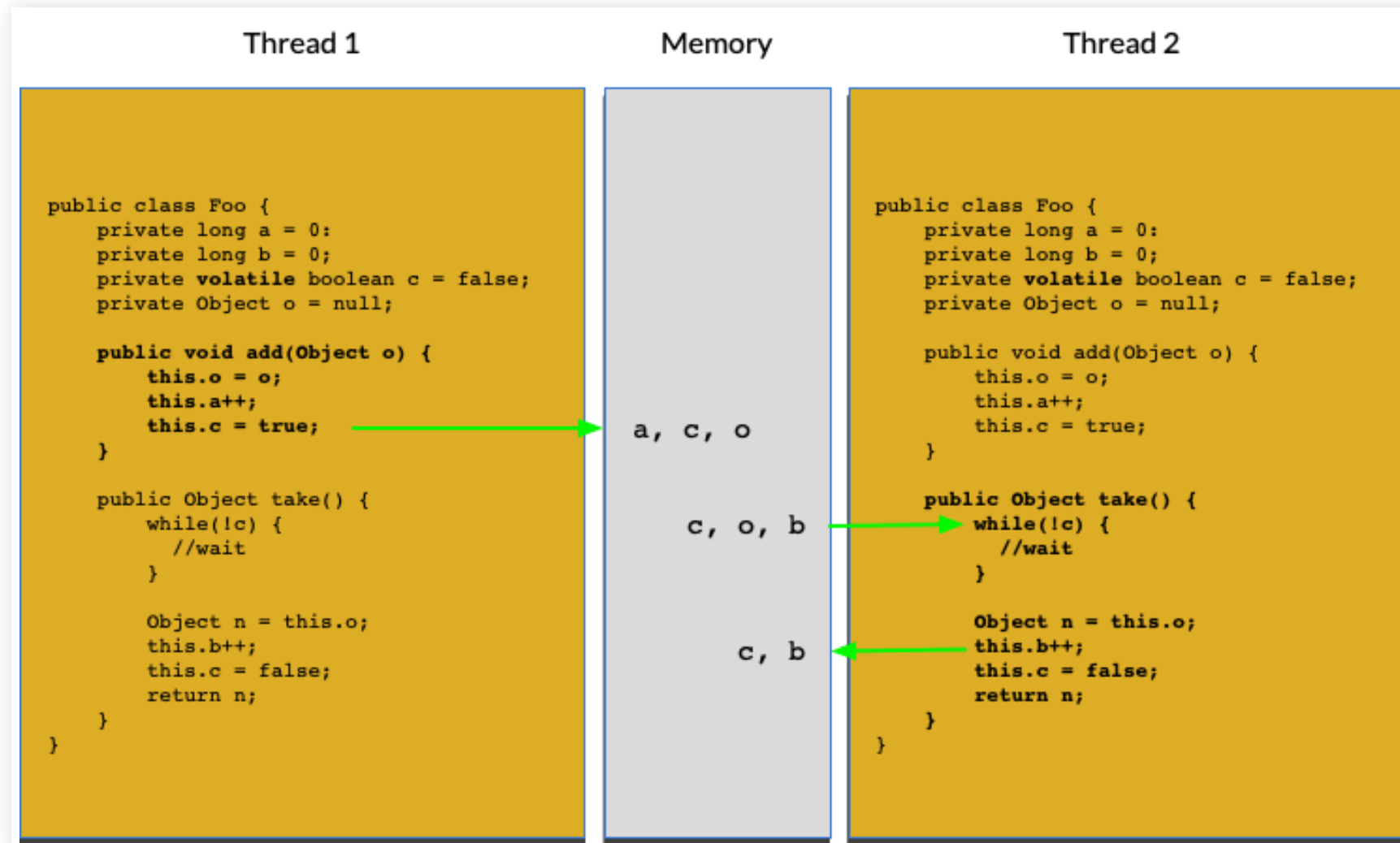


USING THE `volatile` KEYWORD

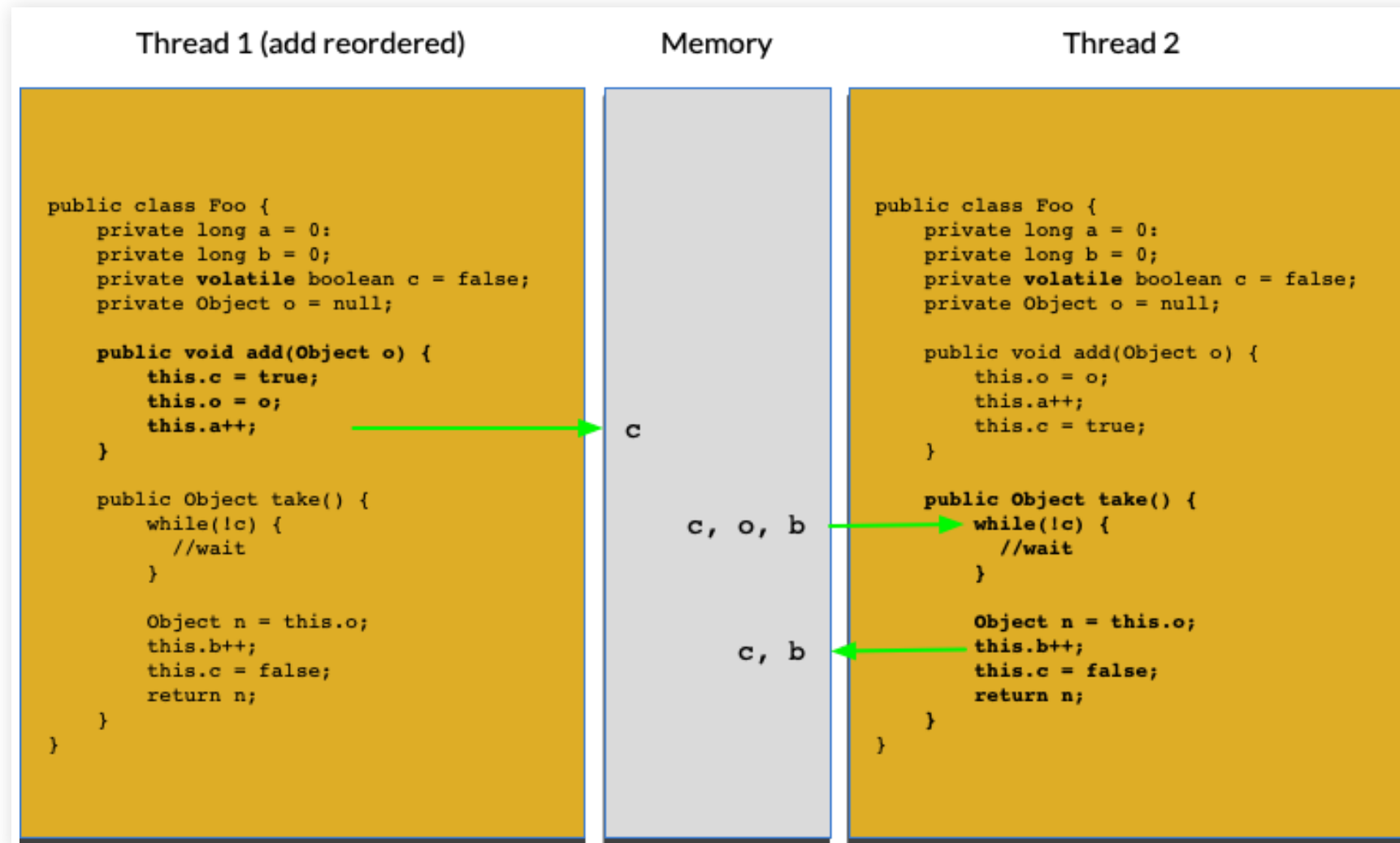
- We can denote each variable as `volatile` but that is unnecessary
- We can select one that will be `volatile`



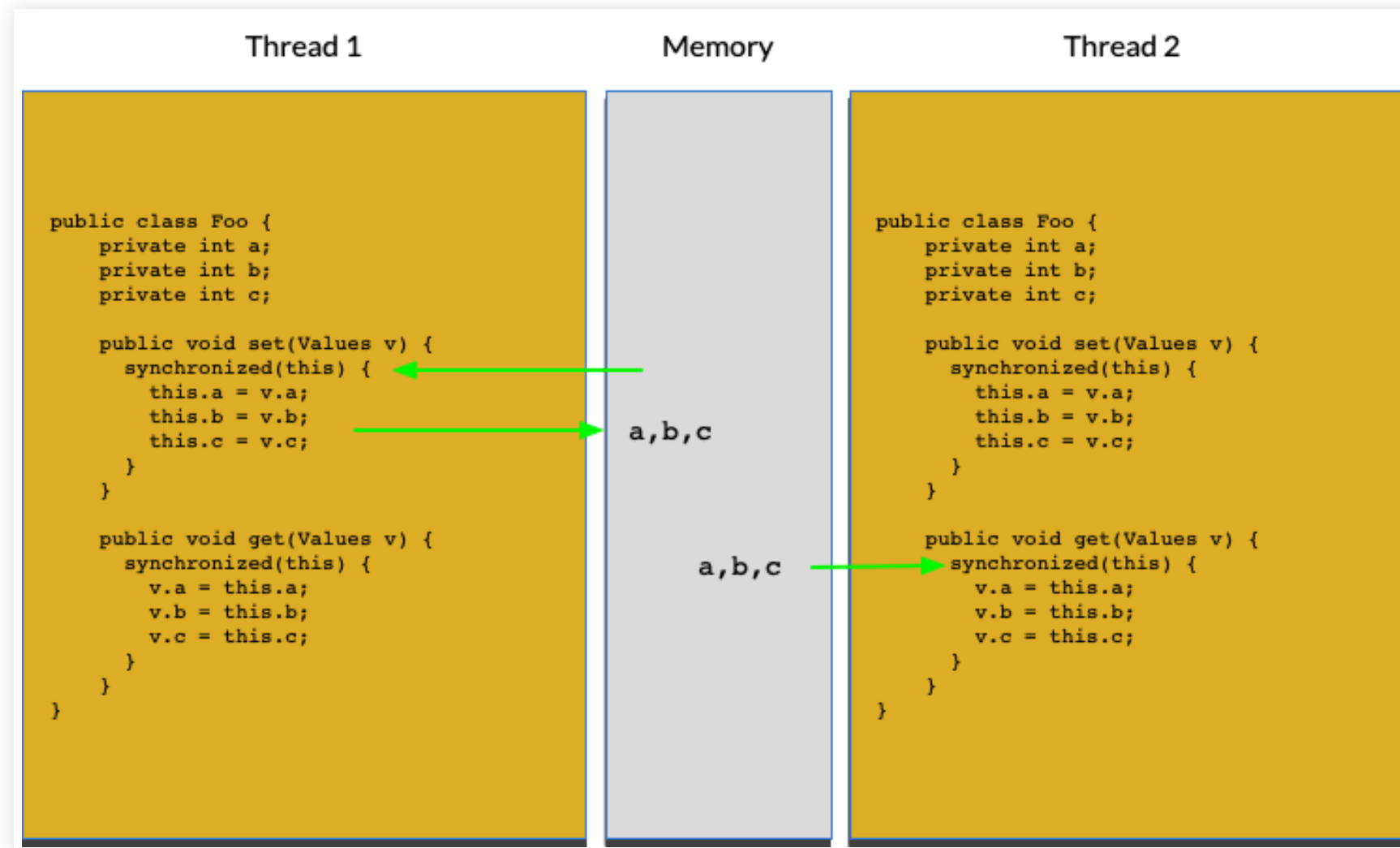
HAPPENS BEFORE GUARANTEE



ORDERING MATTERS



SYNCHRONIZED BLOCKS HAVE AN ORDERING GUARANTEE



IF WE LEAVE VARIABLES OUT OF THE BLOCK

- Variables left out of the block are not guaranteed to be flushed into memory

